Automatic Approximation for the Verification of Cryptographic Protocols

Frédéric Oehl, Gérard Cécé, Olga Kouchnarenko, David Sinclair

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Automatic Approximation for the Verification of Cryptographic Protocols

Frédéric Oehl — Gérard Cécé — Olga Kouchnarenko — David Sinclair

N° 4599

2002, October

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4599--FR+ENG

INRIA
LORRAINE

Automatic Approximation for the Verification of Cryptographic Protocols

Frédéric Oehl* , Gérard Cécé† , Olga Kouchnarenko‡ , David Sinclair§

Thème 2 — Génie logiciel
et calcul symbolique
Projet CASSIS

Rapport de recherche n° 4599 — 2002, October — 18 pages

Abstract: This paper presents an approximation function developed for the verification of cryptographic protocols. The main properties of this approximation are that it can be build automatically and its computation is guaranteed to terminate unlike the Genet and Klay's one. This approximation has been used for the verification of the Needham-Schroeder, Otway-Rees and Woo Lam protocols. To be more precise, the approximation allows us to check secrecy and authenticity properties of the protocols.

Key-words: cryptographic protocols, automatic approximation, verification, secrecy, authenticity property

* School of Computer Applications – Dublin City University, email: foehl@computing.dcu.ie
† LIFC – Université de Franche-Comté, email: cece@univ-fcomte.fr
‡ LIFC – Université de Franche-Comté, email: kouchna@univ-fcomte.fr
§ School of Computer Applications – Dublin City University, email: dsinclai@computing.dcu.ie

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 —Télécopie : +33 3 83 27 83 19

Approximation automatique pour la vérification de protocoles cryptographiques

Résumé : Ce rapport présente une fonction d'approximation qui a été développée pour la vérification de protocoles cryptographiques. Les propriétés principales de cette approximation sont : 1) elle peut être construite automatiquement, 2) son calcul termine, ce qui n'est pas le cas de l'approximation de Genet et Klay. Cette approximation a été utilisée pour vérifier des protocoles de Needham-Schroeder, Otway-Rees and Woo Lam. Plus précisement, cette approximation nous permet de vérifier des propriétés de secret et d'authenticité des protocoles.

Mots-clés : protocoles cryptographiques, approximation automatique, vérification, propriétés de secret et d'authenticité

1 Introduction

Cryptography is used to secure the exchange of information over open networks. *Cryptographic protocols* define the rules (message formats and message order) to establish secure communications. But with some cryptographic protocols, information is not safe even when used with good cryptographic algorithms. So, since these flaws have been discovered in protocols considered to be secure, several methods have been developed to verify cryptographic protocols.

One of the first papers presenting a method to verify cryptographic protocols was [BAN89]. In this paper, Burrows, Abadi and Needham introduce a logic to model and to analyze cryptographic protocols. The idea is to reason about the beliefs of the agents in the network and the evolution of these beliefs after each protocol step. The lack of an automatic tool and of a complete semantics has encouraged the development of other logics [GNY90, AT91].

Existing techniques have also been extended for cryptographic protocol verification. In [Mea94, Mea96], a method based on the model-checking techniques is presented. The technique presents an extension of the Dolev-Yao model [DY83] and also integrates the notion of belief introduced in [BAN89]. The protocol is modeled by sets of rules that describe the intruder abilities, then by a narrowing technique it checks if an insecure state is reachable or not. With the NRL Protocol Analyzer [Mea96] several flaws have been discovered. The main advantage of this tool is that the verification is done automatically. [JRV00] also introduces an automatic tool that has been successfully tested on simple protocols [CJ97]. They use rewriting rules to model the protocol and the intruder behaviour, then they apply those rules with a variant of ac-narrowing on an initial configuration. When the tool found an inconsistency then the protocol is flawed.

In [Pau98], Paulson introduces a method based on the proof by induction to verify cryptographic protocols. This method allows the verification of a large range of properties. But in this approach the secrecy and authenticity properties/theorems are very difficult to prove[1]. The proofs require an experienced user to inject the right lemma at the right time to make the proofs converge. This is not the case of the remaining properties, the proofs of those properties are slightly the same for all protocols.

In [Bol96], Bolignano presents a method based on the clear distinction of reliable and unreliable agents. His method allows a precise specification of the protocol. The properites are modeled with temporal logic features and proved with the help of invariants of the protocol and axioms about the knowledge. The technique has been tested with the Coq prover [Bol95].

The list of techniques to verify cryptographic protocols is long. To have a better view of this particular field of research, the reader can see the surveys [GSG99] and [AGG+01].

Automata and tree automata are well known to model infinite systems. Recently, methods using tree automata to verify cryptographic protocols have been introduced [Mon99, GK00, GL00]. [Mon99] was the first paper where tree automata were used to verify cryptographic protocols. In [Mon99] and [GL00], tree automata model the set of messages that intruders are able to construct. In [GK00] tree automata model the network (traces of the protocol + capabilities of the intruder) and the current intruder knowledge. Another difference between these approaches is that in [Mon99] and [GL00] results are limited to a given number of agents and sessions, which is not the case in [GK00]. To conclude, these methods use an abstract analysis technique to compute the limit (reached when no new information is added to the model)

[1] to see what is involved look at the proofs of the Needham-Schroeder protocol: http://www.cl.cam.ac.uk/Research/HVG/Isabelle/library/HOL/Auth/

after which the computation must stop. To be more precise, the size of the system that must be verified, here a cryptographic protocol, is too big to be explored. To reduce the size of the system and thus be able to verify some properties of this system, an approximated system is built. In this approximated system, only relevant information for the verification are kept. For the techniques previously cited, the approximated system is a super-set of the concrete one. The result is that if a property is satisfied at the end of the computation then the protocol verifies this property, otherwise nothing can be said.

These methods have been used to verify secrecy and authentication properties. Secrecy guarantees that information defined as secret (like shared keys) cannot be caught by an intruder during protocol runs. Authentication guarantees that an agent in the network can identify the sender of a message.

To build an approximated system or an approximation of a system, mathematical functions are used. Those functions, called approximation functions, define which information/parts of the system will be abstracted and how they will be. The title of this paper is "Automatic Approximation ...", so to introduce an automatic approximation the paper first need to present non-automatic one. Thus the paper presents the approximation function of Genet [Gen98] used in [GK00]. Then it gives an automatic (and implemented) way to construct a specific approximation function well adapted for verifying secrecy and authentication on cryptographic protocols. And unlike the Genet and Klay's approximation, this approximation terminates. Secrecy and authentication properties have been checked on the Needham-Schroeder protocol (public key without server, shared key with server), the Woo Lam protocol and the simplified version of Otway-Rees with our approximation function.

The paper is organized as follows. Section 2 introduces some useful definitions. Section 3 presents the approximation of [Gen98] and our approximation. Section 4 explains why our approximation fits to the verification of cryptographic protocols.

2 Definitions

To facilitate the understanding of the rest of the paper some notations and basics definitions are introduced in this section.

Let $\mathcal{F}$ be a finite set of symbols associated with an arity function, $\mathcal{X}$ a countable set of variables, $\mathcal{T}(\mathcal{F},\mathcal{X})$ the set of terms, and $\mathcal{T}(\mathcal{F})$ the set of ground terms (terms without variables). Let $Var(t)$ denote the set of variables of the term $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$.

Definition 1 *A* **term rewriting system** *(TRS) $\mathcal{R}$ is a set of rewrite rules $l \to r$, where $l,\ r \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $l \notin \mathcal{X}$, and $Var(r) \subseteq Var(l)$. If $s|_p$ denotes the subterm of s at the position p and $s[r\sigma]_p$ denotes the term obtained by substitution of the subterm $s|_p$ at the position p by the term $r\sigma$, then the relation $\to_{\mathcal{R}}$ means that for any $s,t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ we have $s \to_{\mathcal{R}} t$ if there exists a rule $l \to r$ in $\mathcal{R}$, a position $p \in \mathcal{P}os(s)$, where $\mathcal{P}os(s)$ is the set of positions in s, and a substitution σ such that $l\sigma = s|_p$ and $t = s[r\sigma]_p$. The set of $\mathcal{R}$-descendants of a set E of ground terms is denoted by $\mathcal{R}^*(E)$ and defined by $\mathcal{R}^*(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E\,.\, s \to_{\mathcal{R}}^* t\}$, where $\to_{\mathcal{R}}^*$ is the transitive closure of $\to_{\mathcal{R}}$.*

Definition 2 *Let $\mathcal{R}$ be a TRS defined on $\mathcal{T}(\mathcal{F},\mathcal{X})$. A term $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ is linear if any variables of $Var(t)$ has exactly one occurrence in t. A rewrite rule is left-linear if the left-hand side of the rule is linear. $\mathcal{R}$ is* **left-linear** *if every rewrite rule of $\mathcal{R}$ is left-linear.*

Definition 3 *A* **bottom-up finite tree automaton** *is a quadruple $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\}$ where $\mathcal{F}$ is a finite set of symbols, $\mathcal{Q}$ is a finite set of states, $\mathcal{Q}_f$ is the set of terminal states such that $\mathcal{Q}_f \subseteq \mathcal{Q}$, and Δ is a set of transitions. A* **transition** *of Δ*

is a rewrite rule $c \rightarrow_{\mathcal{A}} q$, *where* $c \in \mathcal{T}(\mathcal{F} \bigcup \mathcal{Q})$ *and* $q \in \mathcal{Q}$. *The* **tree language** *recognized by* $\mathcal{A}$ *is* $\mathcal{L}(\mathcal{A})=\{t \in \mathcal{T}(\mathcal{F}) \mid \exists q \in \mathcal{Q}_f \, . \, t \rightarrow^*_{\mathcal{A}} q\}$.

>From now on, we consider bottom-up finite tree automata and we say tree automata for short.

3 Approximations

The idea of Genet and Klay is the following. Given an initial automaton $\mathcal{A}$ (recognizing the initial configuration of the network where everybody wants to communicate with everybody), a term rewriting system $\mathcal{R}$ (modeling the protocol steps and the intruder abilities), and an approximation function (see Section 3.1 for more detail), an automaton $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ recognizing an approximation of the possible configurations of the network reachable by $\mathcal{R}$ from $\mathcal{A}$ is built. Moreover, $\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$.

The technique to compute the approximation automaton and the approximation function used by Genet and Klay is explained in the next section. Then our approximation function is introduced and two of its properties are established:

1. $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ computed with our approximation verifies $\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$.
2. The computation of $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ computed with our approximation stops.

3.1 Approximation of Genet and Klay

Let $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\}$ be a tree automaton.

Definition 4 *Given a configuration* $s \in \mathcal{T}(\mathcal{F} \bigcup \mathcal{Q}) \setminus \mathcal{Q}$, *an* **abstraction** *of* s *is a mapping* α*:*

$$\alpha : \{s|_p \mid p \in Pos_{\mathcal{F}}(s)\} \mapsto \mathcal{Q}$$

where $Pos_{\mathcal{F}}(s)$ *denotes the set of positions (sequences of integers) in the term* s. *The mapping* α *is extended on* $\mathcal{T}(\mathcal{F} \bigcup \mathcal{Q})$ *by defining* α *as the identity on* $\mathcal{Q}$.

Definition 5 *Let* $s \rightarrow q$ *be a transition such that* $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, $q \in \mathcal{Q}$, *and* α *an abstraction of* s. *The set* $Norm_\alpha(s \rightarrow q)$ *of normalized transitions is inductively defined by:*

1. *if* $s = q$, *then* $Norm_\alpha(s \rightarrow q) = \emptyset$, *and*
2. *if* $s \in \mathcal{Q}$ *and* $s \neq q$, *then* $Norm_\alpha(s \rightarrow q) = \{s \rightarrow q\}$, *and*
3. *if* $s = f(t_1, \ldots, t_n)$, *then* $Norm_\alpha(s \rightarrow q) =$ $\{f(\alpha(t_1), \ldots, \alpha(t_n)) \rightarrow q\} \cup \bigcup_{i=1}^{n} Norm_\alpha(t_i \rightarrow \alpha(t_i))$.

Definition 6 *Let* $\mathcal{Q}$ *be a set of states,* $\mathcal{Q}_{new}$ *be any set of new states such that* $\mathcal{Q} \bigcap \mathcal{Q}_{new} = \emptyset$, *and* $\mathcal{Q}^*{}_{new}$ *the set of sequences* $q_1 \ldots q_k$ *of states in* $\mathcal{Q}_{new}$. *Let* $\Sigma(\mathcal{Q}, \mathcal{X})$ *be the set of substitutions of variables in* $\mathcal{X}$ *by the states in* $\mathcal{Q}$.

An **approximation function**, γ, *is a triple (a set of rewriting rules, a set of states and a set of substitutions) mapping to a set of sequences of states, i.e.* γ*:* $\mathcal{R} \times (\mathcal{Q} \bigcup \mathcal{Q}_{new}) \times \Sigma((\mathcal{Q} \bigcup \mathcal{Q}_{new}), \mathcal{X}) \mapsto \mathcal{Q}^*{}_{new}$, *such that* $\gamma(l \rightarrow r, q, \sigma) = q_1 \ldots q_k$ *where* $Pos_{\mathcal{F}}(r)$ *is the set of positions in* r *and* $k = Card(Pos_{\mathcal{F}}(r))$.

In the rest of the paper, let $\mathcal{Q}_{new}$ be any set of new states such that $\mathcal{Q} \bigcap \mathcal{Q}_{new} = \emptyset$, and $\mathcal{Q}_u = \mathcal{Q} \bigcup \mathcal{Q}_{new}$.

>From every $\gamma(l \rightarrow r, q, \sigma) = q_1 \ldots q_k$, the states $q_1, \ldots, q_k$ can be associated with positions $p_1, \ldots, p_k \in Pos_{\mathcal{F}}(r)$ by defining the corresponding abstraction

function α on the restricted domain $\{r\sigma|_p \mid \forall\, l \to r \in \mathcal{R},\ \forall p \in \mathcal{P}os_{\mathcal{F}}(r),\ \forall\, \sigma \in \Sigma(\mathcal{Q}, \mathcal{X})\}$: $\alpha(r\sigma|_{p_i}) = q_i$ for all $p_i \in Pos_{\mathcal{F}}(r)$=$\{p_1, \ldots, p_k\}$, such that $p_i < p_{i+1}$ for i=$1 \ldots k$ (where $<$ is the lexicographic ordering). In the following, $Norm_\gamma$ is the normalization function where α value is defined according to γ as above.

Starting from a left-linear TRS $\mathcal{R}$, an initial automaton $\mathcal{A}_0 = \mathcal{A}$ and an approximation function γ, Genet and Klay construct $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$ by:

1. searching for a critical pair, i.e. a state $q \in \mathcal{Q}$, a rewrite rule $l \to r$ and a substitution $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that $l\sigma \to^*_{\mathcal{A}_i} q$ and $r\sigma \twoheadrightarrow^*_{\mathcal{A}_i} q$.
2. $\mathcal{A}_{i+1} = \mathcal{A}_i \cup Norm_\gamma(r\sigma \to q)$.

The above process is iterated until it stops on a tree automaton $\mathcal{A}_k$ such that there is no critical pair.

Before introducing our approximation we give an example that will be re-used later to illustrate the difference between the approximation γ (Definition 6) and our approximation γ_f (Definition 8).

Example 1 Consider the alphabet $\mathcal{F} = \{0 : 0, s : 1\}$, the initial automaton $\mathcal{A}_0 = \{\mathcal{F}, \{q_0, q_1\}, \{q_0, q_1\}, \{0 \to q_0,\ s(q_0) \to q_1\}\}$ and the term rewriting system $\mathcal{R} = \{s(x) \to s(s(x))\}$.

If we apply the above process to compute $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$, the computation will never stop. New states are introduced for each normalization.

The computation without normalization looks like:

$$s(q_0) \to_{\mathcal{R}} s(s(q_0)) \to_{\mathcal{R}} s(s(s(q_0)))...$$

Now if we apply the normalization, we have:

$$s(q_0) \to_{\mathcal{R}} s(q_2) \to_{\mathcal{R}} s(q_3)...$$

with $\gamma(\mathcal{R}, q_1, x = q_0) = q_2$, $\gamma(\mathcal{R}, q_1, x = q_2) = q_3$, ... The computation will go like that for ever!

3.2 Our Approximation Function

In our approach we keep the same normalization as in Definition 5 but we refine the approximation of Genet and Klay (γ; Definition 6), and we have:

Definition 7 *Let $\mathcal{Q}_u^*$ be the set of sequences $q_1 \ldots q_k$ of states in $\mathcal{Q}_u$. Let $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}_u, \mathcal{Q}_f, \Delta\}$ be a tree automaton. Let $Pos_{\mathcal{F}}(r) = \{p_1, \ldots, p_k\}$. An approximation function is a mapping γ_e: $\mathcal{R} \times \mathcal{Q}_u \times \Sigma(\mathcal{Q}_u, \mathcal{X}) \mapsto \mathcal{Q}_u^*$, such that $\gamma_e(l \to r, q, \sigma) = q_1 \ldots q_k$ s.t.*

1. $\forall i.\, i \in [1,k] \Rightarrow ((\exists q'.\, q' \in \mathcal{Q}_u \wedge (r\sigma|_{p_i}) \to^*_{\mathcal{A}} q') \Rightarrow q_i = q')$
2. $\forall i.\, i \in [1,k] \Rightarrow ((\exists f\, .f \in \mathcal{F} \wedge r\sigma = f(t_1, \ldots, t_k) \wedge f(q_1, \ldots, q_k) \to q \wedge (r\sigma|_{p_i}) = f(q_1, \ldots, q_k)) \Rightarrow q_i = q)$
3. $\forall i.i \in [1,k] \Rightarrow (((\exists \sigma', q', l' \to r'.\sigma' \in \Sigma(\mathcal{Q}_u, \mathcal{X}) \wedge q' \in \mathcal{Q}_u \wedge l' \to r' \in \mathcal{R} \wedge \gamma_e(l' \to r', q', \sigma') = q'_1 \ldots q'_z) \Rightarrow (\exists j.\, j \in [1,z] \wedge (r'\sigma'|_{p_j}) = (r\sigma|_{p_i}))) \Rightarrow q_i = q'_j)$.

To facilitate the understanding of Definition 7, we can say:

- the first rule says that if a subterm of $r\sigma$ is already recognized by the state q' of the current automaton then q' is used for the normalization of this subterm;

- the second rule explains that if a subterm of $r\sigma$ at the position p_i is equal to $\gamma_e(l \to r, q, \sigma) = q_1 \dots q_k$ then the state q_i used in $\gamma_e(l \to r, q, \sigma)$ is replaced by the state q;
- the third rule tells that two same subterms in two γ_e have the same normalization state.

We establish now that our function γ_e (Definition 7) gives an upper approximation of $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ and that with this function the computation of the approximation automaton always stops.

Proposition 1 *Let $\mathcal{R}$ be a left-linear TRS. Let γ_e be an approximation function such that γ_e is defined by Definition 7. Let $\mathcal{A}_0$ and $\mathcal{A}_{ek}$ be two tree automata such that $\mathcal{A}_{ek}$ is computed as explained in Section 3.1 with $\mathcal{R}$, γ_e and the initial automaton $\mathcal{A}_0$; Then $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{A}_{ek})$.*

Proof *Proposition 1 is guaranteed by Theorem 1 in [GK00].* ◇

The number of substitutions and of new states is infinite so the computation of the approximated automaton with the function Definition 7 may go forever. To guarantee the termination of the computation, we have to refine Definition 7. The idea is to generate an approximation γ' where only variables that can be replaced by symbols of arity zero are replaced by the states corresponding to those symbols in the tree automaton. Then with the results of γ', the new approximation γ_f is built (Definition 8).

Definition 8 *Let $\mathcal{R}$ be a left-linear term rewriting system. Let $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}_u, \mathcal{Q}_f, \Delta\}$ be a tree automata. Let $Pos_{\mathcal{F}}(r) = \{p_1, \dots, p_k\}$. Let γ' be an approximation function such that γ' is defined by Definition 7 with the substitution $\Sigma(\mathcal{Q}_u, \mathcal{X}) = \bigcup_{i=0}^{Card(E)} \{F_i \mid F_i \subseteq E \wedge Card(F_i) = i\}$ where $E \stackrel{def}{=} \{x = q \,|x \in \mathcal{X} \wedge \exists a.\ (a \in \mathcal{T}(\mathcal{F}) \wedge a \to_{\mathcal{A}} q) \wedge \forall y.\ ((y \in \mathcal{X} \wedge y = x) \Rightarrow y = q \in E)\}$.*

An approximation function γ_f is a mapping γ_f: $\mathcal{R} \times \mathcal{Q}_u \times \Sigma(\mathcal{Q}_u, \mathcal{X}) \mapsto \mathcal{Q}_u^*$ with $\gamma_f(l \to r, q, \sigma) = q_1 \dots q_k$ s.t.

- *$\exists \sigma'.(\sigma' \subseteq \sigma \ .\ \gamma'\ (l \to r, q, \sigma') = q'_1 \dots q'_k)$ and $q_i = q'_i$ $(1 \leq i \leq k)$ with the maximum of matches between σ' and σ,*
- *and:*
  1. *$\forall i.\, i \in [1,k] \ \Rightarrow\ ((\exists j.\, j \in [1,k] \ \wedge\ (r\sigma|_{p_i}) = (r\sigma'|_{p_j}))\ \Rightarrow\ q_i = q'_j)$;*
  2. *$\forall i.\, i \in [1,k] \ \Rightarrow\ ((\exists f.\, f \in \mathcal{F} \ \wedge\ r\sigma' = f(t_1, \dots, t_k) \ \wedge\ f(q'_1, \dots, q'_k) \to q \ \wedge\ (r\sigma|_{p_i}) = f(q'_1, \dots, q'_k))\ \Rightarrow\ q_i = q)$;*
  3. *$\forall i.\, i \in [1,k] \ \Rightarrow\ ((\exists j.\, j \in [1,k] \ \wedge\ (r\sigma|_{p_i}) = (r\sigma|_{p_j}))\ \Rightarrow\ q_i = q_j)$.*

As for Definition 7, we give a brief explanation of the rules of Definition 8:

- the first rule says that if a subterm of $r\sigma$ is already recognized by the state q' of the current automaton then q' is used for the normalization of this subterm;
- the second rule explains that if a subterm of $r\sigma$ at the position p_i is equal to $\gamma_f(l \to r, q, \sigma) = q_1 \dots q_k$ then the state q_i used in $\gamma_f(l \to r, q, \sigma)$ is replaced by the state q;
- the third rule tells that two same subterms of $r\sigma$ have the same normalization state.

This definition of γ_f (Definition 8) also verifies Proposition 1 (Consequence of Theorem 1 in [GK00]).

Proposition 2 *Let $\mathcal{R}$ be a left-linear TRS. Let $\mathcal{A}_{f0} = \{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\}$ be a tree automaton. Let γ_f be an approximation function such that γ_f is defined by Definition 8. If the number of rules in $\mathcal{R}$ and the number of states in $\mathcal{Q}$ are finite then the computation of $\mathcal{A}_{fk}$ converges to a fixpoint.*

Proof *[Sketch]*

- *the first approximation γ' terminates*
  - *initial stage*
    - *Substitutions used in Definition 8 only substitute variables that model symbols of arity 0 by states (the remaining variables can take any values). So the number of substitutions is finite;*
    - *The number of rewriting rules is finite;*
    - *The number of automaton states is finite;*
      - *The numbers of rules $l\sigma \to r\sigma$ with $l \to r \in \mathcal{R}$ and σ a substitution such that $l\sigma \to^*_{\mathcal{A}_{fi}} q$ and $r\sigma \twoheadrightarrow^*_{\mathcal{A}_{fi}} q$ with $q \in \mathcal{Q}$ is finite;*
        If we apply the normalization process to all the previous rules with the function γ' then we add a finite number of states.
  - *n^{th} stage*
    - *So the number of substitutions is finite as the substitution are the same as the ones at the initial stage;*
    - *The number of rewriting rules is finite;*
    - *The number of automaton states is still finite but has been increased of a particular number of states by the last normalization process;*
      - *The numbers of rules $l\sigma \to r\sigma$ with $l \to r \in \mathcal{R}$ and σ a substitution such that $l\sigma \to^*_{\mathcal{A}_{fi}} q$ and $r\sigma \twoheadrightarrow^*_{\mathcal{A}_{fi}} q$ with $q \in \mathcal{Q}$ is finite;*
        If all the 3-uple $\gamma'(l \to r, q, \sigma)$ has been used once, the normalization process does not add any more states to the automaton (because of the substitution used and the definition of γ'); otherwise a finite number of states is added.
  - *After a finite number of normalization processes the computation stops.*
- *the principal approximation γ_f terminates*

  γ_f does not introduce new states, it uses the states introduced by γ' or those of the automaton. So the computation terminates.

$\diamond$

We conclude this section by two examples. The first example was presented in Section 3.1 and the second example, originally from [Gen98], is used to illustrate the utility of γ'. Recall that in [Gen98], the computation of this example does not terminate with the approximation of Definition 6.

Example 2 We have the alphabet $\mathcal{F} = \{0 : 0, s : 1\}$, the initial automaton $\mathcal{A}_0 = \{\mathcal{F}, \{q_0, q_1\}, \{q_0, q_1\}, \{0 \to q_0, s(q_0) \to q_1\}\}$ and the term rewriting system $\mathcal{R} = \{s(x) \to s(s(x))\}$.

If we apply the Genet and Klay process to compute $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$ with our approximation γ_f, the computation stops which is shown below.

The computation without normalization looks like:

$$s(q_0) \to_{\mathcal{R}} s(s(q_0)) \to_{\mathcal{R}} s(s(s(q_0)))...$$

Now if we apply the normalization, we have:

$$s(q_0) \to_{\mathcal{R}} s(q_1) \to_{\mathcal{R}} s(q_2) \to_{\mathcal{R}} s(q_2)$$

We have $\gamma'\ (\mathcal{R}, q_1, x = q_0) = q_1$ and $\gamma'\ (\mathcal{R}, q_1, \emptyset) = q_2$. The first normalization adds no transition ($s(q_0) \to q_1$ is already in $\mathcal{A}_0$). The next normalizations add the transitions $s(q_1) \to q_2$ and $s(q_2) \to q_2$ (because of $\gamma'\ (\mathcal{R}, q_1, \emptyset) = q_2$) and after no more critical pair can be found so the computation stops.

Example 3 In this example taken from [Gen98], we have $\mathcal{A}$ a tree automaton where Δ=$\{app(q_0, q_0) \to q_1, cons(q_2, q_1) \to q_0, nil \to q_0, nil \to q_1, a \to q_2\}$, $rl = app(cons(x, y), z) \to cons(x, app(y, z))$, $\mathcal{R} = \{rl\}$, and γ_f (Definition 8) the approximation function mapping every tuple (rl, q, σ) to one state.

Now, we apply the Genet and Klay process to compute $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$:

1. We have to add $Norm_{\gamma_f}(cons(q_2,\ app(q_1,\ q_0)) \to q_1)$ to Δ.

   $Norm_{\gamma_f}(cons(q_2,\ app(q_1,\ q_0)) \to q_1) = \{cons(q_2, q_3) \to q_1, app(q_1, q_0) \to q_3\}$, to find this set of normalized transitions, we have to compute γ' and then to use the right γ' to have γ_f:

   - γ'=$\begin{cases} \gamma'\ (rl, q_1, \{x = q_2, y = q_1, z = q_0\}) = q_3 \\ \vdots \\ \gamma'\ (rl, q_1, \{x = q_2, z = q_0\}) = q_4 \\ \vdots \\ \gamma'\ (rl, q_1, \emptyset) = q_5 \end{cases}$
   - $\gamma_f\ (rl, q_1, \{x = q_2, y = q_1, z = q_0\}) = \gamma'\ (rl, q_1, \{x = q_2, y = q_1, z = q_0\}) = q_3$; so the transitions $cons(q_2, q_3) \to q_1$ and $app(q_1, q_0) \to q_3$ are added to the current automaton set of transitions.

2. We have to add $Norm_{\gamma_f}(cons(q_2,\ app(q_3,\ q_0)) \to q_3)$ to Δ.

   $Norm_{\gamma_f}(cons(q_2,\ app(q_3,\ q_0)) \to q_3) = \{cons(q_2, q_4) \to q_3, app(q_3, q_0) \to q_4\}$, to find this set of normalized transitions, we repeat the same process as the one in the first step:

   - γ'=$\begin{cases} \gamma'\ (rl, q_3, \{x = q_2, y = q_1, z = q_0\}) = q_3 \\ \vdots \\ \gamma'\ (rl, q_3, \{x = q_2, z = q_0\}) = q_4 \\ \vdots \\ \gamma'\ (rl, q_3, \emptyset) = q_5 \end{cases}$
   - $\gamma_f\ (rl, q_1, \{x = q_2, y = q_3, z = q_0\}) = \gamma'\ (rl, q_1, \{x = q_2, z = q_0\}) = q_4$; so the transitions $cons(q_2, q_4) \to q_3$ and $app(q_3, q_0) \to q_4$ are added to the current automaton set of transitions.

3. We have to add $Norm_{\gamma_f}(cons(q_2,\ app(q_4,\ q_0)) \to q_4)$ to Δ.

   $Norm_{\gamma_f}(cons(q_2,\ app(q_4,\ q_0)) \to q_4) = \{cons(q_2, q_4) \to q_4, app(q_4, q_0) \to q_4\}$, to find this set of normalized transitions, we repeat the same process as the one in the first step:

- $\gamma' = \begin{cases} \gamma'\ (rl, q_4, \{x = q_2, y = q_1, z = q_0\}) = q_3 \\ \vdots \\ \gamma'\ (rl, q_4, \{x = q_2, z = q_0\}) = q_4 \\ \vdots \\ \gamma'\ (rl, q_4, \emptyset) = q_5 \end{cases}$
- $\gamma_f\ (rl, q_4, \{x = q_2, y = q_4, z = q_0\}) = \gamma'\ (rl, q_4, \{x = q_2, z = q_0\}) = q_4$; so the transitions $cons(q_2, q_4) \rightarrow q_4$ and $app(q_4, q_0) \rightarrow q_4$ are added to the current automaton set of transitions.

4. The computation stops and does not go on forever like with the γ function of Genet (Definition 6).

We saw that our approximation is an upper-approximation of what we want to approximate and that the computation of the approximation has stopped.

Now, we can see why this approximation fits to the verification of the cryptographic protocols, and in particular to the verification of the secrecy and authentication properties.

4 Cryptographic Protocols

In this section we explain why our approximation is quite efficient for the verification of cryptographic protocols. We implemented it in OCAML[2] [RV98, LDG+01] to be used by Timbuk[3] [GT01].

In [GK00], the authors explain how cryptographic protocols can be verified with their approximation function (Definition 6). As already said in the introduction, their idea is to compute a superset of all the reachable states (approximation automaton) from the initial configuration of the network, where everybody wants to communicate with everybody (initial automaton), the protocol steps (term rewriting system) and the approximation function. Then the negation of the secrecy and authentication properties are each modeled by a tree automaton (negation automata). The verification of secrecy and authentication are done by checking the intersection of the approximation automaton with the negation automata. If the intersection is empty the property is satisfied, otherwise another technique must be used to verify the property.

To facilitate the understanding of the following comments, the syntax and the semantics used in [GK00] are summarized in Table 1.

The messages exchanged during the protocol runs will be composed of basic pieces of information (i.e. agent name, shared key, ...) or of a concatenation of basic pieces of information (i.e. agent name and shared key encrypted, ...). To reduce the number of messages that can be sent, we decide to fix the format of the messages by typing them. So in the term rewriting system (TRS) instead of having for example $pubkey(x)$ you have $pubkey(agt(x))$ to indicate that x can only be an agent.

In a message, two types of information can be distinguished, the one understood by the agent (i.e. agent names, ...) and the one that cannot be understood by the agent (i.e. an agent cannot access to a piece of information that has been encrypted if he does not have the right decryption key, ...). In TRS, this distinction is visible, if we take the example of a nonce, an agent can identify a nonce if he has created this nonce in the TRS when $agt(x)$ has created a nonce to communicate with $agt(y)$ we have N(agt(x),agt(y)) and when it is a nonce created by someone else we have

[2] http://caml.inria.fr/ocaml/index.html
[3] http://www.irisa.fr/lande/genet/timbuk/index.html

agt(x)	x is an agent
c_init(x, y, z)	x thinks he has established a communication with y but he really communicates with z
c_resp(x, y, z)	x thinks he responds to a request of communication from y but he really communicates with z
cons(x, y)	concatenation of the information x and y
encr(x, y, z)	z is encrypted by y with x
goal(x, y)	x wants to communicate with y
hash1(x, y)	y is hashed by x
hash2(x, y, z)	z is hashed by y with the key x
mesg(x, y, z)	z is a message sent by x to y
N(x, y)	nonce created by x to communicate with y
pubkey(x)	public key of x
serv(x)	x is a server
sharekey(x, y)	key shared by x and y

Table 1: Description of the terms used

N(w,z). Our approximation also makes the distinction, in one case we will have the state corresponding to the precise nonce and in the other case we will introduce a new state (because of the approximation γ'). This approximation γ' gives precise states for known information (as known information contains variable that can be substituted by constant) and abstract states for unknown ones.

The thing to remember is that we want to verify that information during protocol runs are kept secret (secrecy properties) and that at the end the actors of the protocol really communicate with the actors they want to (authentication properties). In [GK00] the authors assume the existence of Alice, Bob (two trustable agents), an unbounded number of untrustable agents and an intruder. They gather together all the untrustable agents to only consider Alice, Bob and the Rest and they verify the secrecy and authentication properties for Alice and Bob. Alice, Bob and the Rest can be seen as constants. In γ', only variables that model them are substituted by a state. The computation of the approximated automaton with this γ' using this substitution has no effect on the verification of our two properties.

To have a better view of what we just said, we can look at the Needham-Schroeder protocol (cf. Fig. 1). Two agents, Alice and Bob, want to establish a secure communication using a public key infrastructure.

	Alice initiates a protocol run, sending a nonce Na and her name A to Bob.
Message 1:	$A \Longrightarrow B : \{Na, A\}_{Kb}$
	Bob responds to Alice's message with a further nonce Nb.
Message 2:	$B \Longrightarrow A : \{Na, Nb, B\}_{Ka}$
	Alice proves her existence by sending Nb back to Bob.
Message 3:	$A \Longrightarrow B : \{Nb\}_{Kb}$

Figure 1: Good version of the Needham-Schroeder protocol

So, for this protocol we verify that the nonce Na (resp. Nb) created by Alice (resp. Bob) to communicate with Bob (resp. Alice) are kept secret during the protocol runs. We also verify that at the end of each run Alice (resp. Bob) really

communicates with Bob (resp. Alice). The approximation function (corresponding to Definition 8) used by Timbuk[4] [GT01] to compute the approximation automaton of Needham-Schroeder is available in Section A. The reader can see that we have all the possible messages, from Alice to Alice, from Alice to Bob, from Alice to someone else, ...

Figure 2 gives the approximation function of the second step of the protocol when Bob, agt(q2), sends the message. The function is composed of rules of the form "[...] -> [...]" where the first part of the rule is the term to normalize and the second one is the normalization process to use. As you can see a precise state is given to information known by Bob, i.e. N(q5, q5) -> q15, and a global state is given to unknown information, i.e. N(a_1, b_1) -> q45 (a_1 and b_1 will be replaced during the computation by q3 (the Rest), q4 (Alice), q5 (Bob)). It is clear on the figure that the nonces created by Bob to communicate with himself, Alice and someone else are not gather together so the verification of the secrecy of the nonce created to communicate with Alice (be sure that the intruder does not catch N(q5, q4)) is not affected by our approximation. For the same reason, distinction of the communication between Alice, Bob and the Rest, our approximation does not affect the verification of the authentication.

```
[U(LHS, mesg(agt(q2), agt(q2), encr(pubkey(agt(q2)), agt(q2), cons(N(q5, q5),
cons(N(agt(q2), agt(q2)), agt(q2)))))) -> q13]
-> [LHS -> q13 agt(q2) -> q5 agt(q2) -> q5 N(q5, q5) -> q15 cons(q15, q5) -> q16
cons(q15, q16) -> q46 pubkey(q5) -> q17 encr(q17, q5, q46) -> q13
mesg(q5, q5, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q2), encr(pubkey(agt(q2)), agt(q2), cons(N(a_1, b_1),
cons(N(agt(q2), agt(q2)), agt(q2)))))) -> q13]
-> [LHS -> q13 agt(q2) -> q5 agt(q2) -> q5 N(q5, q5) -> q15 cons(q15, q5) -> q16
N(a_1, b_1) -> q45 cons(q45, q16) -> q46 pubkey(q5) -> q17 encr(q17, q5, q46) -> q13
mesg(q5, q5, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q1), encr(pubkey(agt(q1)), agt(q2), cons(N(a_2, b_2),
cons(N(agt(q2), agt(q1)), agt(q2)))))) -> q13]
-> [LHS -> q13 agt(q2) -> q5 agt(q1) -> q4 N(q5, q4) -> q19 cons(q19, q5) -> q20
N(a_2, b_2) -> q48 cons(q48, q20) -> q49 pubkey(q4) -> q21 encr(q21, q5, q49) -> q13
mesg(q5, q4, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q0), encr(pubkey(agt(q0)), agt(q2), cons(N(q5, q3),
cons(N(agt(q2), agt(q0)), agt(q2)))))) -> q13]
-> [LHS -> q13 agt(q2) -> q5 agt(q0) -> q3 N(q5, q3) -> q23 cons(q23, q5) -> q24
cons(q23, q24) -> q52 pubkey(q3) -> q25 encr(q25, q5, q52) -> q13
mesg(q5, q3, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q0), encr(pubkey(agt(q0)), agt(q2), cons(N(a_3, b_3),
cons(N(agt(q2), agt(q0)), agt(q2)))))) -> q13]
-> [LHS -> q13 agt(q2) -> q5 agt(q0) -> q3 N(q5, q3) -> q23 cons(q23, q5) -> q24
N(a_3, b_3) -> q51 cons(q51, q24) -> q52 pubkey(q3) -> q25 encr(q25, q5, q52) -> q13
mesg(q5, q3, q13) -> q13]

...
```

Figure 2: Approximation of the second step of Needham Schreoder

[4] http://www.irisa.fr/lande/genet/timbuk/index.html

This distinction is very helpful, when the intersection of the approximation automaton and the negation property automaton is not empty. By looking at the approximation automaton with the approximation function, information, that can help the user to verify whether the property is satisfied or not with another method [Mea96, Pau98, JRV00], can be deduced. In particular, by studying the states of the automaton the user can find which particular step can lead or not to an attack and thus have an idea of how to direct the verification with the other verification technique.

Two comments to conclude this section. First comment, the term rewriting systems used in the protocol case are not inevitably left-linear but it has no consequence for the computation of the approximation. The non-linearity only concerns the agents present in the network and each of them is initially recognized by a precise state. Those states are initially deterministic (you have one state for Alice, one for Bob and one for the Rest) and this property is conserved during the computation (see [GK00] for more detail). Second comment, the examples used in the previous section are not directly related to cryptographic protocols. So it seems that our approximation might also be used in other contexts where you are confronted to an infinite number of reachable states.

5 Conclusion

A tool that generates the approximation function (Definition 8) for protocols without timestamps has been implemented in OCAML[5] [RV98, LDG+01]. This tool also generates the term rewriting system and the initial automaton for the protocol we want to verify. With this tool and the Timbuk library verifications of protocols' properties have been done. The secrecy and authentication have been verified for the Needham-Schroeder protocol (public key without server, shared key with server), the Woo Lam protocol and the simplified version of Otway-Rees.

If we compare Genet and Klay's approximation function with ours, we can say that our function is generated automatically and guarantees to make the computation stop. For the computation time, we cannot compare the results. For our tests we have used Timbuk, which has been specially designed after [GK00] to compute an approximation automaton from an initial automaton, a term rewriting system and an approximation function. It can also be generated automatically before the computation.

Our approximation works well on basic protocols, we are going to test it on more complex protocols (SET [Gro96a], TLS [Gro96b], ...). We will also determine if by extending the semantics of [GK00] it would be possible still with our approximation function to verify freshness properties. Unless most of the other techniques, with [GK00] we can only say that the property is satisfied when the intersection of tree automata is empty. Otherwise we have to use another technique [Mea96, Pau98, JRV00] to verify whether the property is satisfied or not.

In [OS01], we explain how to combine Paulson's idea and the approximation technique to exploit the strengths of each method. This combining approach is illustrated with the Needham-Schroeder protocol in [OS02]. The goal is to develop a technique/tool that could be used by expert in protocols and not in theorem proving.

In the same context of cryptographic protocol verification, we will also look at [BT02] and see how we can adapt their technique to our approach.

The examples solved in this paper with our approximation have no close relation with the cryptographic protocols. We will also look for other fields to successfully

[5] http://caml.inria.fr/ocaml/index.html

use our approximation. May be in the processors/hardware design or algorithm verification...

Acknowledgements Thanks are due to Thomas Genet (Institut de Recherche en Informatique et Systèmes Aléatoires de Rennes), Alain Giorgetti (Laboratoire Informatique de Franche-Comté), Michael Rusinowitch (Laboratoire Lorrain de Recherche en Informatique et ses Applications) and Laurent Vigneron (Laboratoire Lorrain de Recherche en Informatique et ses Applications) for useful comments and discussions. Of course, any mistakes are entirely our responsability.

References

[AGG+01] B. Aziz, D. Gray, G.Hamilton, F. Oehl, J. Power, and D. Sinclair. Implementing protocol verification for e-commerce. *Proceedings of the 2001 International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2001)*, 2001. http://student.dcu.ie/ oehlf2/.

[AT91] Martín Abadi and Mark Tuttle. A semantics for a logic of authentication. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, 1991.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical report, DIGITAL, Systems Research Center, N 39, February 1989. http://www.research.digital.com/SRC/publications/.

[Bol95] D. Bolignano. Vérification formelle de protocoles cryptographiques à l'aide de coq, 1995.

[Bol96] Dominique Bolignano. An approach to the formal verification of cryptographic protocols. In *ACM Conference on Computer and Communications Security*, pages 106–118, 1996.

[BT02] A. Bouajjani and T. Touili. Extrapolating tree transformations, 2002. http://verif.liafa.jussieu.fr/ touili/.

[CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0., 1997.

[DY83] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory, 2(29)*, 1983.

[Gen98] Thomas Genet. Decidable approximations of sets of descendants and sets of normal forms. In *RTA*, pages 151–165, 1998.

[GK00] Thomas Genet and Francis Klay. Rewriting for cryptographic protocol verification. In *CADE: International Conference on Automated Deduction*, 2000. http://citeseer.nj.nec.com/genet99rewriting.html.

[GL00] Jean Goubault-Larrecq. A method for automatic cryptographic protocol verification (extended abstract). In *Proc. Workshop on Formal Methods in Parallel Programming, Theory and Applications (FMPPTA'2000)*, number 1800 in Lecture Notes in Computer Science, pages 977–984. Springer-Verlag, 2000. http://www.dyade.fr/fr/actions/vip/publications.html.

[GNY90] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning About Belief in Cryptographic Protocols. In Deborah Cooper and Teresa Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society, 1990.

[Gro96a] SET Working Group. SET^{TM} specification, books 1,2 and 3. 1996. http://www.setco.org/set_specifications.html.

[Gro96b] TLS Working Group. The TLS protocol version 1.0. 1996. http://www.ietf.org/html.charters/tls-charter.html.

[GSG99] S. Gritzalis, D. Spinellis, and P. Georgiadis. Security protocols over open networks and distributed systems: Formal methods for their analysis, design, and verification. *Computer Communications, 22(8): 695-707*, 1999. http://citeseer.nj.nec.com/gritzalis99security.html.

[GT01] Thomas Genet and Valerie Viet Triem Tong. Reachability analysis of term rewriting systems with timbuk. In *Proc. Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001)*, LNAI. Springer-Verlag, 2001.

[JRV00] Florent Jacquemard, Michael Rusinowitch, and Laurent Vigneron. Compiling and verifying security protocols. *Logic Programming and Automated Reasoning, pages 131-160*, 2000.

[LDG+01] Xavier Leroy, Damien Doligez, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. The objective caml system release 3.02, 2001.

[Mea94] Catherine Meadows. A model of computation for the NRL protocol analyzer. In *CSFW*, 1994.

[Mea96] Cathrine Meadows. The NRL protocol analyser: An overview. *Journal of Logic Programming, 26(2):113–131*, February 1996.

[Mon99] David Monniaux. Abstracting cryptographic protocols with tree automata. In *Static Analysis Symposium*, Lecture Notes in Computer Science, pages 149–163. Springer-Verlag, 1999. http://citeseer.nj.nec.com/monniaux99abstracting.html.

[OS01] F. Oehl and D. Sinclair. Combining two approaches for the verification of cryptographic protocols. *Workshop Specification, Analysis and Validation for Emerging Technologies in Computational Logic (SAVE 2001)*, 2001. http://student.dcu.ie/ oehlf2/.

[OS02] F. Oehl and D. Sinclair. Combining isabelle and timbuk for cryptographic protocol verification. *Workshop Sécurité des Communications sur Internet (SECI 2002)*, 2002. http://student.dcu.ie/ oehlf2/.

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6, 1998. http://www.cl.cam.ac.uk/users/lcp/papers/protocols.html.

[RV98] Didier Rémy and Jérôme Vouillon. Objective ml: An effective object-oriented extension to ml, 1998.

A Approximation of the Needham-Schroeder Protocol

Here we give the automatically generated approximation of the protocol steps for Needham-Schroeder. The state $q1$ models Alice, the state $q2$ models Bob, the state $q0$ models the untrustrable agents of the network and the state $q13$ models the network as in [GK00].

Approximation Needham_Schroeder
States q[0–90]
Rules

The format of the rules is $[U(...)] -> [...]$ *where* $[U(...)]$ *is the term to normalize and* $[...]$ *is the normalization of the first term.*

Approximation of U(LHS, mesg(agt(a), agt(b), encr(pubkey(agt(b)), agt(a), cons(N(agt(a), agt(b)), agt(a)))))

[U(LHS, mesg(agt(q2), agt(q2), encr(pubkey(agt(q2)), agt(q2), cons(N(agt(q2), agt(q2)), agt(q2))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q2) -> q5 N(q5, q5) -> q15 cons(q15, q5) -> q16 pubkey(q5) -> q17 encr(q17, q5, q16) -> q13 mesg(q5, q5, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q1), encr(pubkey(agt(q1)), agt(q2), cons(N(agt(q2), agt(q1)), agt(q2))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q1) -> q4 N(q5, q4) -> q19 cons(q19, q5) -> q20 pubkey(q4) -> q21 encr(q21, q5, q20) -> q13 mesg(q5, q4, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q0), encr(pubkey(agt(q0)), agt(q2), cons(N(agt(q2), agt(q0)), agt(q2))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q0) -> q3 N(q5, q3) -> q23 cons(q23, q5) -> q24 pubkey(q3) -> q25 encr(q25, q5, q24) -> q13 mesg(q5, q3, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q2), encr(pubkey(agt(q2)), agt(q1), cons(N(agt(q1), agt(q2)), agt(q1))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q2) -> q5 N(q4, q5) -> q27 cons(q27, q4) -> q28 pubkey(q5) -> q17 encr(q17, q4, q28) -> q13 mesg(q4, q5, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q1), encr(pubkey(agt(q1)), agt(q1), cons(N(agt(q1), agt(q1)), agt(q1))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q1) -> q4 N(q4, q4) -> q30 cons(q30, q4) -> q31 pubkey(q4) -> q21 encr(q21, q4, q31) -> q13 mesg(q4, q4, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q0), encr(pubkey(agt(q0)), agt(q1), cons(N(agt(q1), agt(q0)), agt(q1))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q0) -> q3 N(q4, q3) -> q33 cons(q33, q4) -> q34 pubkey(q3) -> q25 encr(q25, q4, q34) -> q13 mesg(q4, q3, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q2), encr(pubkey(agt(q2)), agt(q0), cons(N(agt(q0), agt(q2)), agt(q0))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q2) -> q5 N(q3, q5) -> q36 cons(q36, q3) -> q37 pubkey(q5) -> q17 encr(q17, q3, q37) -> q13 mesg(q3, q5, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q1), encr(pubkey(agt(q1)), agt(q0), cons(N(agt(q0), agt(q1)), agt(q0))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q1) -> q4 N(q3, q4) -> q39 cons(q39, q3) -> q40 pubkey(q4) -> q21 encr(q21, q3, q40) -> q13 mesg(q3, q4, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q0), encr(pubkey(agt(q0)), agt(q0), cons(N(agt(q0), agt(q0)), agt(q0))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q0) -> q3 N(q3, q3) -> q42 cons(q42, q3) -> q43 pubkey(q3) -> q25 encr(q25, q3, q43) -> q13 mesg(q3, q3, q13) -> q13]

Approximation of U(LHS, mesg(agt(b), agt(a), encr(pubkey(agt(a)), agt(b), cons(N(a_1, b_1), cons(N(agt(b), agt(a)), agt(b))))))

[U(LHS, mesg(agt(q2), agt(q2), encr(pubkey(agt(q2)), agt(q2), cons(N(q5, q5), cons(N(agt(q2), agt(q2)), agt(q2)))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q2) -> q5 N(q5, q5) -> q15 cons(q15, q5) -> q16 cons(q15, q16) -> q46 pubkey(q5) -> q17 encr(q17, q5, q46) -> q13 mesg(q5, q5, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q2), encr(pubkey(agt(q2)), agt(q2), cons(N(a_1, b_1), cons(N(agt(q2), agt(q2)), agt(q2)))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q2) -> q5 N(q5, q5) -> q15 cons(q15, q5) -> q16 N(a_1, b_1) -> q45 cons(q45, q16) -> q46 pubkey(q5) -> q17 encr(q17, q5, q46) -> q13 mesg(q5, q5, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q1), encr(pubkey(agt(q1)), agt(q2), cons(N(a_2, b_2), cons(N(agt(q2), agt(q1)), agt(q2)))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q1)

-> q4 N(q5, q4) -> q19 cons(q19, q5) -> q20 N(a_2, b_2) -> q48 cons(q48, q20) -> q49 pubkey(q4) -> q21 encr(q21, q5, q49) -> q13 mesg(q5, q4, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q0), encr(pubkey(agt(q0)), agt(q2), cons(N(q5, q3), cons(N(agt(q2), agt(q0)), agt(q2)))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q0) -> q3 N(q5, q3) -> q23 cons(q23, q5) -> q24 cons(q23, q24) -> q52 pubkey(q3) -> q25 encr(q25, q5, q52) -> q13 mesg(q5, q3, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q0), encr(pubkey(agt(q0)), agt(q2), cons(N(a_3, b_3), cons(N(agt(q2), agt(q0)), agt(q2)))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q0) -> q3 N(q5, q3) -> q23 cons(q23, q5) -> q24 N(a_3, b_3) -> q51 cons(q51, q24) -> q52 pubkey(q3) -> q25 encr(q25, q5, q52) -> q13 mesg(q5, q3, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q2), encr(pubkey(agt(q2)), agt(q1), cons(N(a_4, b_4), cons(N(agt(q1), agt(q2)), agt(q1)))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q2) -> q5 N(q4, q5) -> q27 cons(q27, q4) -> q28 N(a_4, b_4) -> q54 cons(q54, q28) -> q55 pubkey(q5) -> q17 encr(q17, q4, q55) -> q13 mesg(q4, q5, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q1), encr(pubkey(agt(q1)), agt(q1), cons(N(q4, q4), cons(N(agt(q1), agt(q1)), agt(q1)))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q1) -> q4 N(q4, q4) -> q30 cons(q30, q4) -> q31 cons(q30, q31) -> q58 pubkey(q4) -> q21 encr(q21, q4, q58) -> q13 mesg(q4, q4, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q1), encr(pubkey(agt(q1)), agt(q1), cons(N(a_5, b_5), cons(N(agt(q1), agt(q1)), agt(q1)))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q1) -> q4 N(q4, q4) -> q30 cons(q30, q4) -> q31 N(a_5, b_5) -> q57 cons(q57, q31) -> q58 pubkey(q4) -> q21 encr(q21, q4, q58) -> q13 mesg(q4, q4, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q0), encr(pubkey(agt(q0)), agt(q1), cons(N(q4, q3), cons(N(agt(q1), agt(q0)), agt(q1)))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q0) -> q3 N(q4, q3) -> q33 cons(q33, q4) -> q34 cons(q33, q34) -> q61 pubkey(q3) -> q25 encr(q25, q4, q61) -> q13 mesg(q4, q3, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q0), encr(pubkey(agt(q0)), agt(q1), cons(N(a_6, b_6), cons(N(agt(q1), agt(q0)), agt(q1)))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q0) -> q3 N(q4, q3) -> q33 cons(q33, q4) -> q34 N(a_6, b_6) -> q60 cons(q60, q34) -> q61 pubkey(q3) -> q25 encr(q25, q4, q61) -> q13 mesg(q4, q3, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q2), encr(pubkey(agt(q2)), agt(q0), cons(N(a_7, b_7), cons(N(agt(q0), agt(q2)), agt(q0)))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q2) -> q5 N(q3, q5) -> q36 cons(q36, q3) -> q37 N(a_7, b_7) -> q63 cons(q63, q37) -> q64 pubkey(q5) -> q17 encr(q17, q3, q64) -> q13 mesg(q3, q5, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q1), encr(pubkey(agt(q1)), agt(q0), cons(N(a_8, b_8), cons(N(agt(q0), agt(q1)), agt(q0)))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q1) -> q4 N(q3, q4) -> q39 cons(q39, q3) -> q40 N(a_8, b_8) -> q66 cons(q66, q40) -> q67 pubkey(q4) -> q21 encr(q21, q3, q67) -> q13 mesg(q3, q4, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q0), encr(pubkey(agt(q0)), agt(q0), cons(N(q3, q3), cons(N(agt(q0), agt(q0)), agt(q0)))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q0) -> q3 N(q3, q3) -> q42 cons(q42, q3) -> q43 cons(q42, q43) -> q70 pubkey(q3) -> q25 encr(q25, q3, q70) -> q13 mesg(q3, q3, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q0), encr(pubkey(agt(q0)), agt(q0), cons(N(a_9, b_9), cons(N(agt(q0), agt(q0)), agt(q0)))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q0) -> q3 N(q3, q3) -> q42 cons(q42, q3) -> q43 N(a_9, b_9) -> q69 cons(q69, q43) -> q70 pubkey(q3) -> q25 encr(q25, q3, q70) -> q13 mesg(q3, q3, q13) -> q13]

Approximation of U(LHS, mesg(agt(a), agt(b), encr(pubkey(agt(b)), agt(a), N(a_2, b_2))))

[U(LHS, mesg(agt(q2), agt(q2), encr(pubkey(agt(q2)), agt(q2), N(a_10, b_10)))) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q2) -> q5 N(a_10, b_10) -> q73 pubkey(q5) -> q17 encr(q17, q5, q73) -> q13 mesg(q5, q5, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q1), encr(pubkey(agt(q1)), agt(q2), N(a_11, b_11)))) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q1) -> q4 N(a_11, b_11) -> q75 pubkey(q4) -> q21 encr(q21, q5, q75) -> q13 mesg(q5, q4, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q0), encr(pubkey(agt(q0)), agt(q2), N(a_12, b_12)))) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q0) -> q3 N(a_12, b_12) -> q77 pubkey(q3) -> q25 encr(q25, q5, q77) -> q13 mesg(q5, q3, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q2), encr(pubkey(agt(q2)), agt(q1), N(a_13, b_13)))) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q2) -> q5 N(a_13, b_13) -> q80 pubkey(q5) -> q17 encr(q17, q4, q80) -> q13 mesg(q4, q5, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q1), encr(pubkey(agt(q1)), agt(q1), N(a_14, b_14)))) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q1) -> q4 N(a_14, b_14) -> q82 pubkey(q4) -> q21 encr(q21, q4, q82) -> q13 mesg(q4, q4, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q0), encr(pubkey(agt(q0)), agt(q1), N(a_15, b_15)))) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q0) -> q3 N(a_15, b_15) -> q84 pubkey(q3) -> q25 encr(q25, q4, q84) -> q13 mesg(q4, q3, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q2), encr(pubkey(agt(q2)), agt(q0), N(a_16, b_16)))) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q2) -> q5 N(a_16, b_16) -> q86 pubkey(q5) -> q17 encr(q17, q3, q86) -> q13 mesg(q3, q5, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q1), encr(pubkey(agt(q1)), agt(q0), N(a_17, b_17)))) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q1) -> q4 N(a_17, b_17) -> q88 pubkey(q4) -> q21 encr(q21, q3, q88) -> q13 mesg(q3, q4, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q0), encr(pubkey(agt(q0)), agt(q0), N(a_18, b_18)))) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q0) -> q3 N(a_18, b_18) -> q90 pubkey(q3) -> q25 encr(q25, q3, q90) -> q13 mesg(q3, q3, q13) -> q13]

(* End of the function *)

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399